Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Кафедра параллельных алгоритмов

Сазонова Галина Олеговна

Адаптивное сплайн-всплесковое разложение дискретного числового потока

Выпускная квалификационная работа

Научный руководитель:
д. ф.-м. н., профессор Демьянович Ю. К.

Рецензент:
кандидат ф.-м. н., доцент Лебединский Д. М.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics and Mechanics Faculty
SubDepartment of Parallel Algorithms

Sazonova Galina Olegovna

Adaptive spline-wavelet decomposition of the discrete digital flow

Graduate qualification work

Scientific supervisor:
Doctor of Science, Professor Dem'yanovich Yu. K.

Reviewer:
Ph. D, Assoc. professor Lebedinsky D. M.

Saint-Petersburg
2017

Оглавление

Введение

В настоящее время приходится иметь дело с потоками цифровой информации внушительного объема. Для их быстрой обработки необходимы достаточно большие компьютерные ресурсы, потому вопрос о сокращении объемов информации за счет исключения несущественных составляющих является актуальным.

Среди средств решения данного вопроса лидируют вейвлеты. Вейвлетное разложение, как правило, рассматривается на равномерной сетке (например, [1, 6]). Однако, в последнее время стали распространяться сплайн-всплесковые разложения, связанные с адаптивной сеткой (например, [3, 7]). Неравномерные сетки важны в случае неравномерного поведения потока: при медленном изменении – крупная сетка, в областях быстрого изменения – мелкая. При таком подходе возможно последовательное адаптивное укрупнение возникающих таким образом неравномерных сеток для получения всплескового пакета с заданной аппроксимацией исходного потока.

Цель данной работы состоит в компьютерном моделировании адаптивного сплайн-всплескового разложения исходного числового потока. В работе дана программная реализация построения адаптивной сетки, учитывающей поведение исходного потока, а также реализованы алгоритмы декомпозиции и реконструкции. Программные реализации упомянутых алгоритмов применены для обработки звуковых файлов в формате WAVE.

Работа содержит введение, три главы, заключение, список литературы и приложение.

В первой главе описывается сплайн-всплесковое разложение дискретных потоков (см. [5]): приведена основная идея основного подхода, описан процесс процесс построения адаптивной сетки, представлены матрицы сужения и продолжения.

Во второй главе отражена алгоритмическая сторона реализации адаптивной сетки и процессов декомпозиции и реконструкции в сплайн-всплесковом разложении. Третья глава посвящена программной реа-

лизации упомянутого разложения; здесь дано описание программы и представлены результаты ее использования для обработки wav-файлов.

В приложении приведены фрагменты программы. Полный текст программы доступен по ссылке https://github.com/GalinaSazonova.

1. Сплайн-всплесковое разложение

1.1. Основная идея разложения. Неклассический подход

Для начала приведем здесь общую идею сплайн-всплескового разложения на одномерном примере (см. [3]). В качестве области определения рассмотрим интервал (α, β) вещественной оси.

Рассмотрим сетку

$$X : \ldots < x_{-2} < x_{-1} < x_0 < x_1 < x_2 < \ldots, \tag{1.1.1}$$

$$\lim_{j \to -\infty} x_j = \alpha, \qquad \lim_{j \to -\infty} x_j = \beta, \tag{1.1.2}$$

и вектор-функцию $\varphi(t) = (\varphi_0(t), \varphi_1(t), \ldots, \varphi_m(t))$, $t \in (\alpha, \beta)$, $\varphi_i \in \mathbb{L}$, $i \in \{0, 1, \ldots, m\}$, где $\mathbb{L}$ – линейное пространство функций, определенных на интервале (α, β).

Введем множество линейных функционалов $G = \{g^{(s)}\}_{s \in \mathbb{Z}}$, $g^{(s)} \in \mathbb{L}^*$, со свойством

$$\operatorname{supp} g^{(s)} \subset (x_s, x_{s+1}) \qquad \forall s \in \mathbb{Z}. \tag{1.1.3}$$

Результат действия функционала $g^{(s)}$ на функцию $u \in \mathbb{L}$ обозначается острыми скобками $\langle g^{(s)}, u \rangle$; далее приведено представление результата действия функционала на вектор-функцию $\varphi(t)$:

$$\langle g^{(s)}, \varphi \rangle = (\langle g^{(s)}, \varphi_0 \rangle, \langle g^{(s)}, \varphi_1 \rangle, \ldots, \langle g^{(s)}, \varphi_m \rangle)^T.$$

Предположим, что выполнено условие

$$\det(\langle g^{(s)}, \varphi \rangle, \langle g^{(s+1)}, \varphi \rangle, \ldots, \langle g^{(s+m)}, \varphi \rangle) \neq 0 \qquad \forall s \in \mathbb{Z}. \tag{1.1.4}$$

Положим

$$a_s = \langle g^{(s)}, \varphi \rangle. \tag{1.1.5}$$

Векторы $a_s, a_{s+1}, \ldots, a_{s+m+1}$ линейно независимы по условию (1.1.4).

Теперь рассмотрим аппроксимационные соотношения

$$\sum_{i=k-m}^{k} a_i \omega_i(t) = \varphi(t) \qquad \forall t \in (x_k, x_{k+1}) \qquad \forall k \in \mathbb{Z}, \tag{1.1.6}$$

$$\operatorname{supp} \omega_s \subset [x_s, x_{s+m+1}] \qquad \forall s \in \mathbb{Z}. \tag{1.1.7}$$

Из формул (1.1.6) – (1.1.7) однозначно определяются функции $\omega_i(t)$.

Предположим, что $\omega_s \in \mathbb{L}$. Согласно формулам (1.1.3) – (1.1.7) имеем $\langle g^{(j)}, \omega_s \rangle = \delta_{j,s} \quad \forall j, s \in \mathbb{Z}$. Введем линейное пространство $\mathbb{S} = \mathbb{S}(X, \varphi) = \mathcal{L}\{\omega_s\}_{s \in \mathbb{Z}}$, где $\mathcal{L}$ — линейная оболочка.

Если рассмотреть подмножество $\widetilde{X}$ сетки X такое, что

$$\widetilde{X} : \ldots < \widetilde{x}_{-2} < \widetilde{x}_{-1} < \widetilde{x}_0 < \widetilde{x}_1 < \widetilde{x}_2 < \ldots,$$

$$\lim_{j \to -\infty} \widetilde{x}_j = \alpha, \qquad \lim_{j \to -\infty} \widetilde{x}_j = \beta, \qquad \widetilde{X} \subset X,$$

то можно найти функции $\widetilde{\omega}_i$, связанные с новой сеткой $\widetilde{X}$ и определить линейное пространство $\widetilde{\mathbb{S}} = \mathbb{S}(\widetilde{X}, \varphi)$, которое окажется подпространством пространства $\mathbb{S}(X, \varphi)$ при определенных условиях.

Введем в рассмотрение оператор P, который проектирует пространство $\mathbb{S}$ на подпространство $\widetilde{\mathbb{S}}$:

$$Pu = \sum_{s \in \mathbb{Z}} \langle \widetilde{g}^{(s)}, u \rangle \omega_s \qquad \forall u \in \mathbb{S}(X, \varphi), \tag{1.1.8}$$

где $\{\widetilde{g}^{(s)}\}_{s \in \mathbb{Z}}$ — фиксированная система функционалов, биортогональная системе функций $\{\widetilde{\omega}_i\}_{i \in \mathbb{Z}}$. Если $t \in (\widetilde{x}_k, \widetilde{x}_{k+1})$ фиксировано, то правая часть формулы (1.1.8) имеет не более $m + 1$ слагаемого:

$$Pu(t) = \sum_{s=k-m}^{k} \langle \widetilde{g}^{(s)}, u \rangle \omega_s(t) \qquad \forall t \in (\widetilde{x}_k, \widetilde{x}_{k+1}). \tag{1.1.9}$$

Проектирующий оператор P определяет всплесковое разложение

$$\mathbb{S} = \widetilde{\mathbb{S}} + \mathbb{W}. \tag{1.1.10}$$

Пусть $c = (\ldots, c_{-2}, c_{-1}, c_0, c_1, c_2, \ldots)$ – исходный поток числовой ин-

формации. Рассмотрим функцию

$$u(t) = \sum_j c_j \omega_j(t); \tag{1.1.11}$$

ее проекция $\widetilde{u} = Pu$ на пространство $\widetilde{\mathbb{S}}$ может быть представлена в форме

$$\widetilde{u} = \sum_i a_i \widetilde{\omega}_i. \tag{1.1.12}$$

В итоге имеем основной поток $a = (\ldots, a_{-2}, a_{-1}, a_0, a_1, a_2, \ldots)$, который соответствует укрупнению $\widetilde{X}$ сетки X, а также вспесковый поток $b = (\ldots, b_{-2}, b_{-1}, b_0, b_1, b_2, \ldots)$, который определяется разложением разности $w = u - \widetilde{u}$ по базису пространства $\mathbb{S}$: $w = \sum_s b_s \omega_s$ (см. формулы (1.1.9) – (1.1.12)).

Переход от исходного потока c к потокам a и b называется *декомпозицией*, а обратный переход называется *реконструкцией*. Формулы декомпозиции могут быть представлены в форме $a = \mathfrak{Q}c$, $b = c - \mathfrak{P}^T \mathfrak{Q}c$, а формулы реконструкции – в форме $c = b + \mathfrak{P}^T a$, где $\mathfrak{P}$ и $\mathfrak{Q}$ – матрицы сужения и продолжения соответственно. Если отрезок $[a, b]$ содержится в интервале (α, β), то все предыдущие построения справедливы для сужения рассматриваемых функций на этот отрезок; при этом сетки, а также исходный, основной и всплесковый потоки оказываются конечными.

Далее будут рассматриваться сеточные функции, областью определения которых является сетка вида (1.1.1) – (1.1.2) или ее часть. Такой подход обусловлен тем, что он удобен при работе с числовыми потоками.

Обратимся к [5], чтобы вывести формулы декомпозиции и реконструкции для дискретного случая.

1.2. Укрупнение сетки

Пусть на интервале (α, β) рассматривается сетка

$$\Xi: \quad \ldots < \xi_{-2} < \xi_{-1} < \xi_0 < \xi_1 < \xi_2 \ldots, \tag{1.2.1}$$

$$\lim_{i\to-\infty}\xi_i=\alpha,\quad \lim_{i\to+\infty}\xi_i=\beta. \tag{1.2.2}$$

$C(\Xi)$ – множество функций $u(t)$, заданных на сетке Ξ.

Обозначим $a^-=\xi_{i-1}$, $a^+=\xi_{i+1}$.

Предполагается, что $a,b\in\Xi,\quad a^+<b^-$, и тогда вводится обозначение $[\![a,b]\!]=\{\xi_s\mid a\le\xi_s\le b,\ s\in\mathbb{Z}\}$, т.е. $[\![a,b]\!]=\{\xi_s\mid i\le s\le j,\ s\in\mathbb{Z}\}$; множество $[\![a,b]\!]$ называется сеточным отрезком. В дальнейшем условимся считать, что при $c>d$ множество $[\![c,d]\!]$пусто.

Рассматривается линейное нормированное пространство $C[\![a,b]\!]$ функций $u(t)$, заданных на сеточном отрезке $[\![a,b]\!]$, где норма вводится соотношением

$$\|u\|_{C[\![a,b]\!]}=\max_{t\in[\![a,b]\!]}|u(t)|.$$

Пространство $C[\![a,b]\!]$ конечномерное.

Далее для натурального числа m положим

$$J_m=\{0,1,\dots,m\},\qquad J'_m=\{-1,0,1,\dots,m\}.$$

На сеточном отрезке $[\![a,b]\!]$, $a=\xi_0<\xi_1<\ldots<\xi_{M-1}<\xi_M=b$, рассмотрим функции $\{\omega_j(t)\}_{j\in J'_{M-1}}$ пространства $C[\![a,b]\!]$:

$$\omega_j(\xi_s)=\delta_{s,j+1},\qquad s\in J_M,$$

а также линейные функционалы $g^{(i)}$, $i\in J'_{M-1}$ вида

$$\langle g^{(i)},u\rangle=u(\xi_{i+1})\qquad\forall u\in C[\![a,b]\!].$$

Система $\{\omega_j\}_{j\in J'_{M-1}}$ является базисом в пространстве $C[\![a,b]\!]$; она называется дискретным базисом.

Пусть $5\le K<M$. Рассмотрим инъективное отображение $\varkappa$ множества J_K в множество J_M, при котором

$$\varkappa(0)=0,\quad \varkappa(i)<\varkappa(i+1),\quad \varkappa(K)=M.$$

Введем множество

$$J^*=\varkappa J_K,\quad J^*\subset J_M.$$

На этом множестве определено однозначное обратное отображение

$$\forall r \in J^* \quad \varkappa^{-1} : r \longrightarrow s, \quad s \in J_K, \quad J_K = \varkappa^{-1} J^*.$$

Рассмотрим новую сетку

$$\widehat{X} : \quad a = \widehat{x}_0 < \widehat{x}_1 < \ldots < \widehat{x}_K = b,$$

где $\widehat{x}_i = \xi_{\varkappa(i)}$, $i \in J_K$.

Введем функции $\widehat{\omega}_j(t)$, $j \in J'_{K-1}$:

$$\widehat{\omega}_i(t) = (t - \xi_{\varkappa(i)})(\xi_{\varkappa(i+1)} - \xi_{\varkappa(i)})^{-1}$$

$$\text{при} \quad t \in [\![\xi^{+}_{\varkappa(i)}, \xi_{\varkappa(i+1)}]\!], \; i \in J_{K-1},$$

$$\widehat{\omega}_i(t) = (\xi_{\varkappa(i+2)} - t)(\xi_{\varkappa(i+2)} - \xi_{\varkappa(i+1)})^{-1}$$

$$\text{при} \quad t \in [\![\xi_{\varkappa(i+1)}, \xi^{-}_{\varkappa(i+2)}]\!], \; i \in J'_{K-2};$$

$$\widehat{\omega}_i(t) = 0 \quad \text{при} \quad t \in [\![a, b]\!] \setminus [\![\xi^{+}_{\varkappa(i)}, \xi^{-}_{\varkappa(i+2)}]\!].$$

Ясно, что

$$\widehat{\omega}_i(\xi_{\varkappa(i+1)}) = 1 \quad \forall i \in J'_{K-1}.$$

1.3. Матрицы сужения и продолжения

Сплайны $\widehat{\omega}_i$ могут быть представлены в виде линейных комбинаций сплайнов ω_j:

$$\widehat{\omega}_i(t) = \sum_{j \in J'_{M-1}} \mathfrak{p}_{i,j} \omega_j(t) \quad \forall t \in [\![a, b]\!], \quad i \in J'_{K-1},$$

называемых *калибровочными соотношениями.*

Матрица, называемая матрицей сужения, имеет вид $\mathfrak{P} = (\mathfrak{p}_{i,j})$ $i \in J'_{K-1}, j \in J'_{M-1}$, где

$$\mathfrak{p}_{i,j} = \langle g^{(j)}, \widehat{\omega}_i \rangle.$$

Введем упорядоченные (по возрастанию) подмножества множества целых чисел

$$J^0 = \{-1, \ldots, \varkappa(1) - 2\},$$

$$J^1(r) = \{\varkappa(r), \ldots, \varkappa(r+1) - 1\} \qquad \forall r \in J_{K-1},$$

$$J^2(r) = \{\varkappa(r+1), \ldots, \varkappa(r+2) - 2\} \qquad \forall r \in J_{K-2},$$

$$J(r) = J^1(r) \bigcup J^2(r) \quad \forall r \in J_{K-2}, \qquad J(K-1) = J^1(K-1).$$

Условимся считать множество пустым, если первое из выписанных чисел больше последнего.

Следующие утверждения доказаны в [5].

Теорема 1. *Справедливы калибровочные соотношения*

$$\widehat{\omega}_r(t) = \sum_{q \in J'_{M-1}} \mathfrak{p}_{r,q} \omega_q(t) \quad \forall t \in [\![a, b]\!], \qquad r \in J'_{K-1}, \tag{1.3.1}$$

где

$$\mathfrak{p}_{-1,q} = \frac{\xi_{\varkappa(1)} - \xi_{q+1}}{\xi_{\varkappa(1)} - \xi_{\varkappa(0)}} \qquad q \in J^0, \tag{1.3.2}$$

$$\mathfrak{p}_{r,q} = \frac{\xi_{q+1} - \xi_{\varkappa(r)}}{\xi_{\varkappa(r+1)} - \xi_{\varkappa(r)}} \qquad q \in J^1(r), \quad r \in J_{K-1}, \tag{1.3.3}$$

$$\mathfrak{p}_{r,q} = \frac{\xi_{\varkappa(r+2)} - \xi_{q+1}}{\xi_{\varkappa(r+2)} - \xi_{\varkappa(r+1)}} \qquad q \in J^2(r), \quad r \in J_{K-2}; \tag{1.3.4}$$

а неупомянутые в формулах (1.3.2) – (1.3.4) элементы $\mathfrak{p}_{r,q}$ *матрицы* $\mathfrak{P}$ *равны нулю.*

Следствие 1. *Калибровочные соотношения (1.3.1) – (1.3.4) для* $\forall t \in [\![a, b]\!]$ *могут быть представлены в виде*

$$\widehat{\omega}_{-1}(t) = \sum_{j \in J^0} \mathfrak{p}_{-1,j} \omega_j(t), \tag{1.3.5}$$

$$\widehat{\omega}_i(t) = \sum_{j \in J(i)} \mathfrak{p}_{i,j} \omega_j(t) \qquad i \in J'_{K-2}, \tag{1.3.6}$$

$$\widehat{\omega}_{K-1}(t) = \sum_{j \in J^1(K-1)} \mathfrak{p}_{K-1,j} \omega_j(t) \quad \forall t \in [\![a, b]\!], \tag{1.3.7}$$

а также в виде

$$\widehat{\omega}_{-1}(t) = \sum_{j \in J^0} \frac{\xi_{\varkappa(1)} - \xi_{j+1}}{\xi_{\varkappa(1)} - \xi_{\varkappa(0)}} \omega_j(t), \tag{1.3.8}$$

$$\widehat{\omega}_i(t) = \sum_{j\in J^1(i)} \frac{\xi_{j+1} - \xi_{\varkappa(i)}}{\xi_{\varkappa(i+1)} - \xi_{\varkappa(i)}}\omega_j(t) + \sum_{j\in J^2(i)} \frac{\xi_{\varkappa(i+2)} - \xi_{j+1}}{\xi_{\varkappa(i+2)} - \xi_{\varkappa(i+1)}}\omega_j(t),$$

$$i \in J_{K-2}, \tag{1.3.9}$$

$$\widehat{\omega}_{K-1}(t) = \sum_{j\in J^1(K-1)} \frac{\xi_{j+1} - \xi_{\varkappa(K-1)}}{\xi_{\varkappa(K)} - \xi_{\varkappa(K-1)}}\omega_j(t) \quad \forall t \in [\![a, b]\!]. \tag{1.3.10}$$

Далее рассмотрим простые утверждения, приведенные в [5], касающиеся матрицы продолжения. Сначала положим $\mathfrak{q}_{s,j} = \langle \widehat{g}^{(s)}, \omega_j \rangle$, и рассмотрим матрицу $\mathfrak{Q} = (\mathfrak{q}_{s,j})_{s\in J'_{K-1}, j\in J'_{M-1}}$, называемую *матрицей продолжения.*

Утверждение 1. *Справедливы следующие утверждения:*

1) нулевыми являются те столбцы $\mathfrak{q}^{(j)} = (\mathfrak{q}_{sj})_{s\in J'_{K-1}}$ *матрицы* $\mathfrak{Q}$*, номер* j *которых удовлетворяет условию* $j + 1 \notin J^*$*;*

2) остальные столбцы, т.е. столбцы, номер j *которых удовлетворяет условию* $j + 1 \in J^*$*, содержат единицу на месте* s_0*, где* $\varkappa(s_0 + 1) = j + 1$*; остальные элементы* j*-го столбца равны нулю.*

Утверждение 2. *Матрица* $\mathfrak{Q}$ *является левой обратной к матрице* $\mathfrak{P}^T$*:*

$$\mathfrak{Q}\mathfrak{P}^T = I,$$

где I *— единичная матрица размеров* $K + 1 \times K + 1$*.*

1.4. Формулы декомпозиции и реконструкции

Введем в рассмотрение три вектора

$$a = (a_{-1}, a_0, a_1, a_2, \ldots, a_{K-1}),$$

$$b = (b_{-1}, b_0, b_1, b_2, \ldots, b_{M-1}),$$

$$c = (c_{-1}, c_0, c_1, c_2, \ldots, c_{M-1}),$$

назовем их основным, всплесковым и исходным числовыми потоками соответственно.

Запишем формулы декомпозии и реконструкции в матричном виде, используя построения, предложенные в [2], вообще говоря относящиеся

к всплесковому разложению для функций на континууме, но подходящие и для рассматриваемого дискретного случая:

$$c = b + \mathfrak{P}^T a,$$

$$a = \mathfrak{Q} c, \qquad b = c - \mathfrak{P}^T \mathfrak{Q} c.$$

Теорема 2. *Для формул декомпозиции справедливы соотношения*

$$a_i = c_{\varkappa(i+1)-1} \qquad \forall i \in J'_{K-1}, \tag{1.4.1}$$

$$b_q = 0 \qquad \forall q+1 \in J^*, \tag{1.4.2}$$

$$b_q = c_q - \sum_{j \in J'_{K-1}} \langle g^{(q)}, \widehat{\omega}_j \rangle c_{\varkappa(j+1)-1} \quad \forall q+1 \in J_M \backslash J^*. \tag{1.4.3}$$

Теорема 3. *Для всплескового потока при $q+1 \in J_M \backslash J^*$ верны равенства*

$$b_q = c_q - (\widehat{x}_{s+1} - \widehat{x}_s)^{-1} \Big[(\widehat{x}_{s+1} - \xi_{q+1}) c_{\varkappa(s)-1} + (\xi_{q+1} - \widehat{x}_s) c_{\varkappa(s+1)-1} \Big], \tag{1.4.4}$$

где

$$\widehat{x}_s < \xi_{q+1} < \widehat{x}_{s+1}. \tag{1.4.5}$$

1.5. Адаптивная сетка

До этого момента укрупненная сетка априори была задана отображением $\varkappa$. Теперь же рассмотрим алгоритм адаптивного укрупнения заданной сетки. Пусть дана сетка (1.2.1)– (1.2.2).

Пусть для функции $f \in C(\Xi)$ и для некоторой константы $c > 0$ справедливо

$$f(t) \geq c \quad \forall t \in \Xi. \tag{1.5.1}$$

.

Пусть

$$\varepsilon \in (\varepsilon^*, \varepsilon^{**}), \tag{1.5.2}$$

где

$$\varepsilon^* = \max_{\xi \in [\![a,\, b^-]\!]} \max_{t \in \{\xi, \xi^+\}} f(t)(\xi^+ - \xi), \quad \varepsilon^{**} = (b-a)\|f\|_{C[\![a,\, b]\!]}. \tag{1.5.3}$$

Лемма 1. *Если выполнены условия (1.5.1) – (1.5.3), то существуют и единственны натуральное число* $K = K(f, \varepsilon, \Xi)$ *и сетка*

$$\widetilde{X} = \widetilde{X}(f, \varepsilon, \Xi): \quad a = \widetilde{x}_0 < \widetilde{x}_1 < \ldots < \widetilde{x}_K \le \widetilde{x}_{K+1} = b \tag{1.5.4}$$

такие, что

$$\max_{t \in [\![\widetilde{x}_s,\, \widetilde{x}_{s+1}]\!]} f(t)(\widetilde{x}_{s+1} - \widetilde{x}_s) \le \varepsilon < \max_{t \in [\![\widetilde{x}_s,\, \widetilde{x}^+_{s+1}]\!]} f(t)(\widetilde{x}^+_{s+1} - \widetilde{x}_s)$$

$$\forall s \in \{0, 1, \ldots, K-1\}, \tag{1.5.5}$$

$$\max_{t \in [\![\widetilde{x}_K,\, b]\!]} f(t)(b - \widetilde{x}_K) \le \varepsilon, \quad \widetilde{X} \subset \Xi. \tag{1.5.6}$$

Лемма доказывается индукционным переходом от узлов $\widetilde{x}_1, \ldots, \widetilde{x}_s$ к узлам $\widetilde{x}_1, \ldots, \widetilde{x}_{s+1}$.

Сетка, удовлетворяющая условиям леммы называется сеткой адаптивного вида для дискретной функции f со свойством (1.5.1).

2. Алгоритмическая реализация сплайн-всплескового разложения

Вычисление сплайн-всплескового разложения включает в себя 2 этапа:

1) реализация формул декомпозиции; этот этап содержит две подзадачи: отыскание основного потока и отыскание всплескового потока;

2) реализация формул реконструкции.

Отыскание основного потока является более важной задачей, потому что именно всплесковый поток дает понимание характера исходного потока.

2.1. Построение адаптивной сетки

Построение адаптивной сетки происходит с использованием результатов, описанных в п. 1.5. Числовой поток характеризуется сеточной функцией $u_j, j \in \{1, \dots R+2\}$. Рассмотрим случай, когда исходная сетка равномерная, а именно $\xi_j = jh$, $h > 0$; тогда соотношения (1.5.5) – (1.5.6) запишутся в виде:

$$\max_{t \in [\![j_s h, j_{s+1} h]\!]} f(t)(j_{s+1} - j_s)h \le \varepsilon < \max_{t \in [\![j_s h, (j_{s+1}+1) h]\!]} f(t)(j_{s+1} + 1 - j_s)h$$

$$\forall s \in \{0, 1, \dots, K-1\}, \tag{2.1.1}$$

$$\max_{t \in [\![j_{Kh}, j_{K+1} h]\!]} f(t)(j_{K+1} - j_K)h \le \varepsilon,\ j_{K+1} = R,\ j_i = \frac{\xi_i}{h}. \tag{2.1.2}$$

При этом соотношения (1.5.3) примут вид:

$$\varepsilon^* = ||f||_{C[\![a,b]\!]} h, \quad \varepsilon^{**} = (b-a)||f||_{C[\![a,b]\!]},$$

что можно записать как

$$\varepsilon^* = h \max_{j \in \{0,1,\dots,R+1\}} f(jh), \quad \varepsilon^{**} = (b-a) \max_{j \in \{0,1,\dots,R+1\}} f(jh). \tag{2.1.3}$$

Первым шагом алгоритма отыскания сетки является проверка корректности переданного ε; если оно удовлетворяет условию (2.1.3), то приступаем к построению сетки. Последовательно обходим узлы, проверяя соотношение

$$f(jh) = |u_{j+1} - u_j| \leq \varepsilon, \tag{2.1.4}$$

которое гарантирует выполение условий (2.1.1) – (2.1.2). Такой процесс приведет к адаптивной сетке, зависящей от исходного числового потока. Результатом работы программы является массив, хранящий узлы этой сетки.

Реализована функция, которая, используя сплайны первого порядка, восстанавливает исходный поток по ранее построенной сетке.

Построение адаптивной сетки является частью реализации формул декомпозиции и реконструкции, а именно, полученные узлы сетки определяют отображение $\varkappa$. Эмпирическим путем было установлено, что значения исходного потока в узлах адаптивной сетки определяют основной поток (см. также лемму 2 в работе [4]).

Основной недостаток предложенного алгоритма состоит в том, что построение – преимущественно последовательный процесс.

2.2. Реализация формул декомпозиции и реконструкции

Напомним обозначения для векторов исходного, основого и всплескового потоков:

$$a = (a_{-1}, a_0, a_1, a_2, \ldots, a_{K-1}),$$
$$b = (b_{-1}, b_0, b_1, b_2, \ldots, b_{M-1}),$$
$$c = (c_{-1}, c_0, c_1, c_2, \ldots, c_{M-1}),$$

Рассмотрим сначала реализацию декомпозиции. Для этого перепишем формулы декомпозиции (1.4.1) – (1.4.2):

$$a_i = c_{\varkappa(i+1)-1} \qquad \forall i \in J'_{K-1}, \tag{2.2.1}$$

$$b_q = 0 \qquad \forall q+1 \in J^*, \tag{2.2.2}$$

соотношения (1.4.4) – (1.4.5) можно переписать в виде

$$b_q = c_q - (\xi_{\varkappa(i+1)} - \xi_{\varkappa(i)})^{-1}\Big[(\xi_{\varkappa(i+1)} - \xi_{q+1})c_{\varkappa(i)-1} + \\ +(\xi_{q+1} - \xi_{\varkappa(i)})c_{\varkappa(i+1)-1}\Big] \forall q+1 \in J\backslash J^*, \tag{2.2.3}$$

где

$$\varkappa(i)+1 \le q+1 \le \varkappa(i+1)-1. \tag{2.2.4}$$

Так как основной поток определяется значениями исходного потока, которые соответствуют узлам построенной ранее адаптивной сетки, построение основного и всплескового потоков будет проходить таким образом: последовательно просматривается множество J, которое соответвует сетке исходного потока; те значения, которые совпадают со значениями из множества J^*, определяют основной поток по формуле (2.2.1), а также обнуляют соответвующие значения всплеского потока согласно формуле (2.2.2). Значения из множества $J\backslash J^*$ определяют всплесковый поток по формуле (2.2.3). Главный вопрос состоит в реализации формулы (2.2.4); это означает, что нам необходимо отыскать такое значение i, чтобы значение $q \in J\backslash J^*$ находилось между двумя ближайшими значениями из множества J^*. Данное вычисление лучше вынести в отдельную функцию.

В следующей теореме получим формулы реконструкции.

Теорема 4. *Для формул реконструкции справедливы соотношения*

$$c_q = a_q \qquad \forall q+1 \in J^*, \tag{2.2.5}$$

$$c_q = b_q + \sum_{j \in J'_{K-1}} \langle g^q, \widehat{\omega}_j \rangle a_j \qquad \forall q+1 \in J_M \backslash J^*. \tag{2.2.6}$$

Доказательство. Формулы реконструкции (2.2.5) – (2.2.6) получаются элементарными преобразованиями из формул (1.4.1) – (1.4.3). ■

Используя формулы (1.3.8) – (1.3.10), формулы реконструкции (2.2.5)

– (2.2.6) перепишем в виде:

$$c_q = a_q \qquad \forall q+1 \in J^*, \tag{2.2.7}$$

$$c_q = b_q + (\xi_{\varkappa(i+1)} - \xi_{\varkappa(i)})^{-1}\Big[(\xi_{\varkappa(i+1)} - \xi_{q+1})a_i + (\xi_{q+1} - \xi_{\varkappa(i)})a_{i+1}\Big], \tag{2.2.8}$$

где

$$\varkappa(i) + 1 \le q + 1 \le \varkappa(i+1) - 1. \tag{2.2.9}$$

Перейдем к рассмотрению реализации формул реконструкции. Процесс реконструкции устроен аналогично процессу декомпозиции, то есть также последовательно просматривается множество J и согласно формулам (2.2.5) – (2.2.6) восстанавливается исходный поток; так как значительных отличий в организации процесса нет, перейдем к рассмотрению самой программы и результатов ее работы.

3. Программная реализация сплайн-всплескового разложения

3.1. Описание программы

Программа реализована на алгоритмическом языке C^{++}. Основная логика программы содержится в классе, в котором реализованы все упомянутые выше функции. При инициализации класса автоматически определяется ε как треть от ε^{**}; при желании можно поменять это значение, для этого реализована специальная функция; новое ε будет принято, только в случае удовлетворения условию (2.1.3).

Построение адаптивной сетки осуществляется в функции $grid$, результатом работы являются два массива: $adnetindex$ хранит узлы полученной сетки, $adnetres$ хранит значения, соответвующие этим узлам, т.е. основной поток. Текст данной функции приведен в Приложении 1.

Теперь обратимся к функции восстановления с использованием адаптивной сетке; в данной программе она носит название $agrid$. Восстановленный исходный поток сохраняется в массив $flowrestired$. Процесс восстановления проходит последовательно с помощью сплайнов первого порядка. По соседним значениям на адаптивной сетке строится сплайн и с его помощью находятся значения, опущенные при укрупнении исходной сетки. Соответствующая этому алгоритму функция приведена в Приложении 1.

Всплесковый поток необходим для уточнения восстанавливаемого потока; часто для понимания характера исходного потока достаточно основного потока. Ввиду этого формулы декомпозиции и реконструкции реализованы в отдельных функциях.

Формулы декомпозиции (2.2.2) – (2.2.4) реализованы в функции $makeWaveFlow$. Согласно п.2.2. реализация формулы (2.2.4)

$$\varkappa(i) + 1 \leq q + 1 \leq \varkappa(i+1) - 1$$

вынесена в отдельную функцию под названием $findKappaIndex$; ее текст приведен в Приложении 1. Вызов этой функции происходит перед

тем, как вычислить очередное отличное от нуля значение очередного b_q.

Всплесковый поток сохраняется в массиве $waveflow$. Работа упомянутой функции состоит в последовательном просмотре индексов множества J (натуральных чисел от 1 до размера массива с исходными данными $adnetstop$). Если значение индекса совпадает со значением индекса из адаптивной сетки, значение b_q обнуляется; в противном случае происходит вызов функции $findKappaIndex$ и вычисление b_q по формуле

$$b_q = c_q - (\xi_{\varkappa(i+1)} - \xi_{\varkappa(i)})^{-1}\Big[(\xi_{\varkappa(i+1)} - \xi_{q+1})c_{\varkappa(i)-1} + \\ +(\xi_{q+1} - \xi_{\varkappa(i)})c_{\varkappa(i+1)-1}\Big] \quad \forall q+1 \in J\backslash J^*.$$

При реализации формул реконструкции необходима функция $findKappaIndex$. Для восстановления используются потоки $waveflow$ и $adnetres$. Функция реконструкции носит название $rebuildFlow$ и включает в себя последовательный процесс обхода индексов массива $waveflow$. Восстановление происходит с использованием формул (2.2.7) – (2.2.9)

$$c_q = a_q \qquad \forall q+1 \in J^*, \tag{3.1.1}$$

$$c_q = b_q + (\xi_{\varkappa(i+1)} - \xi_{\varkappa(i)})^{-1})\Big[(\xi_{\varkappa(i+1)} - \xi_{q+1})a_i + (\xi_{q+1} - \xi_{\varkappa(i)})a_{i+1}\Big], \tag{3.1.2}$$

где

$$\varkappa(i) + 1 \le q + 1 \le \varkappa(i+1) - 1. \tag{3.1.3}$$

Результат восстановления записывается в массив $flowrestored2$, если индекс совпадает с индексом основного потока, то значение из основного потока просто вставляется в восстановленный поток (3.1.1). Если индекс не совпадает, то происходит вызов фукции $findKappaIndex$ (то есть идет поиск значения q по формуле (3.1.3)) и значение посчитывается по (3.1.2).

В Приложении 1 можно ознакомиться с текстами обеих функций.

3.2. Результат работы программы

В качестве дискретного потока было решено взять дискретизацию звука, так как она представляет собой набор семплов, взятых в определенные промежутки времени, то есть это числовые данные на равномерной сетке с шагом равным частоте дискретизации.

Ввиду известной теоремы Котельникова гладкого изменения исходного (звукового) потока ожидать не приходится (ибо частоту дискретизации аналогового сигнала берут небольшой). Таким образом, тестирование программы проводится на негладких исходных потоках.

Программа принимает на вход звуковой файл в формате WAVE. После считывания данные хранятся в виде динамического массива формата short (это обусловлено тем, что чаще всего под семпл выделяется 16 бит). Из-за выбранного формата работа ведется с целочисленным дискретным потоком. Кроме функции чтения звукового файла написана функция записи данных в новый файл типа wav, для того, чтобы была возможность сравнить звучание исходного файла со звучанием после его обработки и восстановления.

Ознакомимся с результатами работы программы. Реализовано два пути к восстановлению файла: восстановление по адаптивной сетке (по основному потоку) и восстановление с помощью формул реконструкции (задействованы и основной, и всплесковый потоки). Точность восстановления зависит от величины ε, от укрупнения сетки и характера потока (при резкой смене характера потока желательно использовать малое ε).

3.2.1. Реконструкция с использованием адаптивной сетки

Сначала рассмотрим восстановление с использованием лишь основного потока. При этом у нас есть только сетка и значения потока в ее узлах. Далее приведены графики для конкретного wav-файла, иллюстрирующие фрагмент исходного потока (синий цвет) и его версию, полученную восстановлением с использованием адаптивной сетки (красный цвет). Для наглядности рассмотрим несколько вариантов значений ε,

ε^{**} вычисляется согласно формуле (2.1.3):

$$\varepsilon^{**} = (b-a) \max_{j \in \{0,1,\ldots,R+1\}} f(jh).$$

Рисунок 1 соответствует случаю $\varepsilon = \varepsilon^{**}/12$, в данном случае $\varepsilon = 465$. По рисунку видно, что аппроксимация получилась достаточно точной; к тому же при прослушивании данного аудио разница с исходным файлом неразличима. На рис.2 используем $\varepsilon = \varepsilon^{**}/3, \quad \varepsilon = 1860$, на рис.3 рассмотрим $\varepsilon = \varepsilon^{**}/2, \quad \varepsilon = 2790$ и на рис.4 берем $\varepsilon = \varepsilon^{**} - 100, \quad \varepsilon = 5480$. Очевидно, что с увеличением ε происходит укрупнение сетки и соответственно падает точность восстановления исходного потока. Восстановленный звук дает четкое понятие об исходном звуке, легко различить, что произносится, но при больших ε появляются механические шумы на фоне.

Для каждого восстановленного потока (т.е. для каждого ε) можно вычислить среднее уклонение от исходного потока, сложив все значения абсолютных разностей между исходным и восстановленным потоками и поделив на их количество, которое совпадает с размерами исходного и восстановленного потоков. Для $\varepsilon = 465$, соответвующего первому рисунку, среднее уклонение равно $115,9$. Для $\varepsilon = 1860$ это отклонение равно $687,6$, для $\varepsilon = 2790$ оно равно $1100,9$, для $\varepsilon = 5480$ среднее уклонение $2048,5$.

Теперь рассмотрим, какой объем информации занимает хранение адаптивной сетки и значений в ее узлах относительно объема исходного файла. Рассмотрим таблицу 1, каждая строка соответствует новому wav-файлу, по столбцам таблица поделена на 3 основных блока:

1. Информация в первом блоке соответствует исходному файлу, в столбцах этого блока указывается объем памяти, выделяемой на диске, и количество семплов (размер потока), характеризующих звук.

2. Второй блок содержит информацию о файле, хранящем основной поток. Для него указывается точность аппроксимации (т.е. значение ε, вычисляемое по формуле указанной в таблице, в данном

случае $\varepsilon = \varepsilon^{**}/12$), количество элементов в сетке (размер сетки) и объем памяти, выделяемой на диске [1].

3. Третий блок содержит информацию о коэффициенте сжатия исходного файла, который вычисляется как отношение размера исходного файла к размеру файла, хранящего основной поток.

Таблицы 2, 3, 4 устроены аналогично, каждая таблица построена для разных значений ε. Можно заметить, что с ростом ε растет и коэффциент сжатия, соответственно уменьшается объем файла, хранящего адаптивную сетку. Для большей наглядности результат наблюдений помещен в таблицу 5, значения соответвуют первому рассматриваему wav-файлу каждой таблицы (первая строка), в таблице размещены значения ε (точность аппроксимации) и коэффциенты сжатия.

Таблица 1

Исх. файл		Точность аппрокс. ($\varepsilon^{**}/12$)			Коэф. сжатия
Кб	Р-р потока	ε	Кб	Р-р сетки	(исх. / рез.)
93	47268	465	78	13285	$1,2$
90	45620	424	76	12922	$1,2$
222	113511	244	183	31227	$1,2$

Таблица 2

Исх. файл		Точность аппрокс. ($\varepsilon^{**}/3$)			Коэф. сжатия
Кб	Р-р потока	ε	Кб	Р-р сетки	(исх. / рез.)
93	47268	1860	30	4983	$3,1$
90	45620	1697	33	5548	$2,7$
222	113511	976	38	6457	$5,8$

[1] В новый файл сохраняется заголовок wav-файла (всегда 44 байта), номера узлов сетки (по 4 байта на каждый, т.к. хранятся в формате int) и значения в этих узлах (по 2 байта на каждое значение, т.к. хранятся в формате short).

Таблица 3

Исх. файл		Точность аппрокс. ($\varepsilon^{**}/2$)			Коэф. сжатия
Кб	Р-р потока	ε	Кб	Р-р сетки	(исх. / рез.)
93	47268	2790	18	3035	$5,1$
90	45620	2546	21	3511	$4,2$
222	113511	1464	22	3589	10

Таблица 4

Исх. файл		Точность аппрокс. ($\varepsilon^{**} - 100$)			Коэф. сжатия
Кб	Р-р потока	ε	Кб	Р-р сетки	(исх. / рез.)
93	47268	5480	6	1005	$15,5$
90	45620	4993	7	1134	$12,8$
222	113511	2829	8	1329	$27,7$

Таблица 5

Точность аппрокс.(ε)	Коэф. сжатия
465	$1,2$
1860	$3,1$
2790	$5,1$
5480	$15,5$

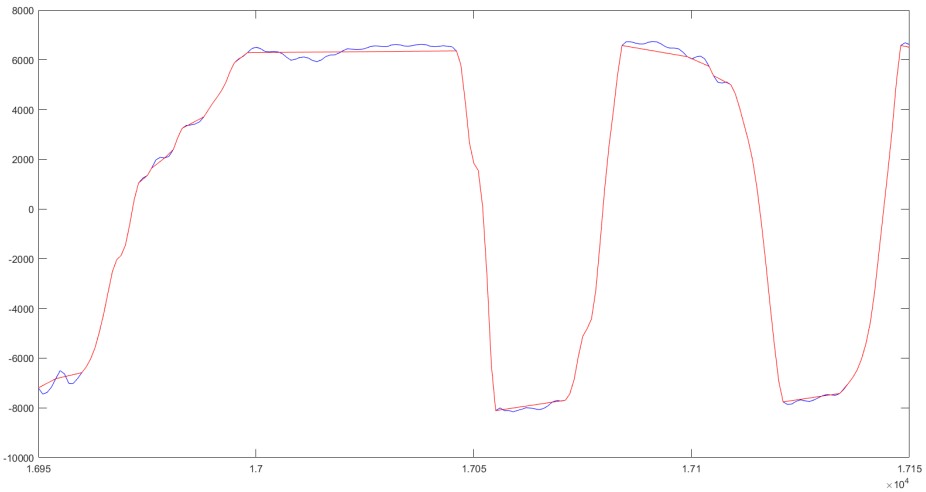

Рис. 1

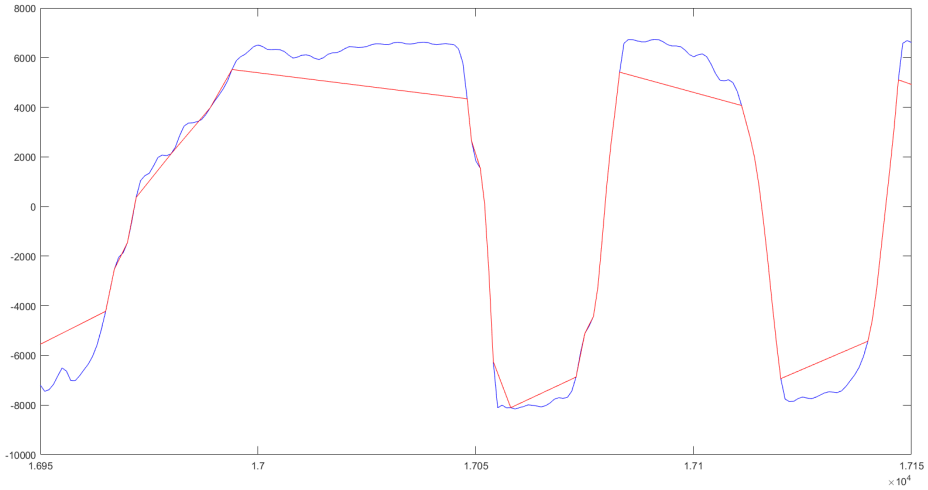
8000
6000
4000
2000
0
-2000
-4000
-6000
-8000
-10000
1.695
1.7
1.705
1.71
1.715
×10^4

Рис. 2

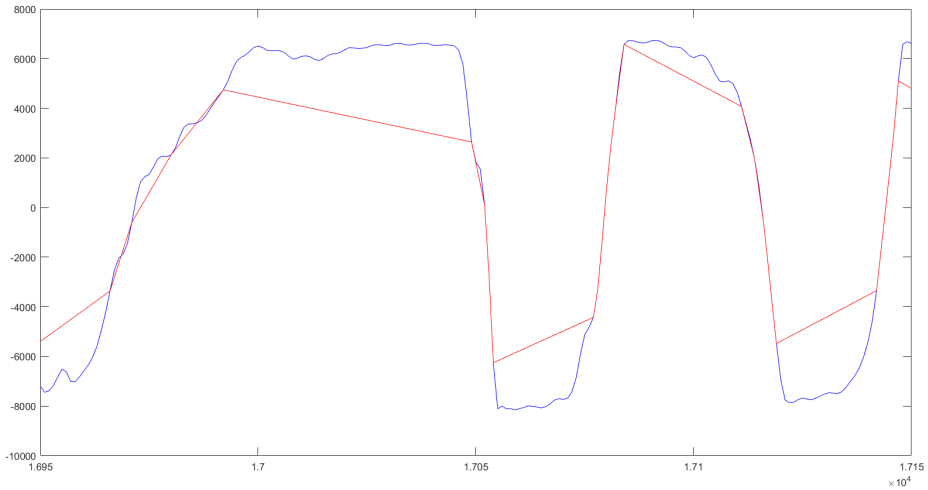
8000
6000
4000
2000
0
-2000
-4000
-6000
-8000
-10000
1.695
1.7
1.705
1.71
1.715
×10^4

Рис. 3

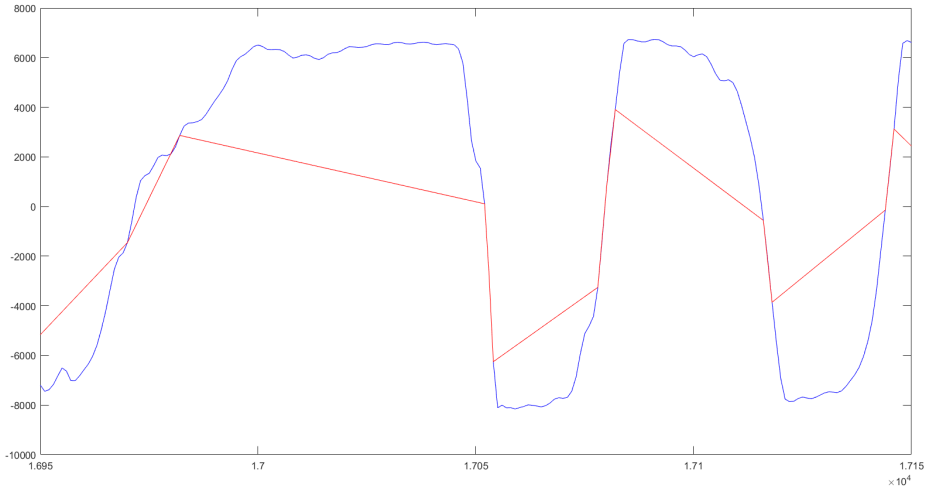

Рис. 4

3.2.2. Результат применения формул реконструкции

Теперь внимательнее рассмотрим восстановление при помощи формул реконструкции, то есть с использованием всплескового потока. Предназначение этого потока состоит в уточнении основного потока для того, чтобы иметь возможность максимально точно восстановить исходный поток. Для демонстрации точности была реализована функция, которая считает отношение полностью совпавших значений в массивах, хранящих исходный поток и восстановленный (массив $flowrestored2$, где хранится поток, восстановленный по формулам реконструкции), ко всему количеству семплов в звуковом файле. Данная функция была применена к различным потокам, обработанным с использованием нескольких значений ε (аналогичных предыдущему случаю).

Средний результат совпадения потоков – $98,87\%$. [2] Среднее значение отклонения потоков равно $0,0151$.

[2]Формулы реконструкции должны давать полное совпадение, но результат получился не точным, потому что исследовались целочисленные потоки и погрешность приходится на округления при вычислениях с плавающей точкой.

Заключение

В данной работе были рассмотрены основные понятия, связанные со сплайн-всплесковым разложением первого порядка. Были составлены алгоритмы реализации процессов декомпозиции и реконструкции, которые затем были программно реализованы. Работа полученного приложения была продемострирована в случае обработки звукового сигнала. По результатам работы приложения были сделаны выводы относительно качества восстановления.

По материалам данной работы были написаны статьи "Построение адаптивной сетки для дискретного потока" (см. [8]) и "Построение адаптивной сетки для обработки звукового сигнала"(в печати), которые были представлены на конференциях "Процессы управления и устойчивость-2016" и "СПИСОК-2017" соответственно.

Список литературы

[1] Terekhov K., Vassilevski Yu. Two-phase water flooding simulations on dynamic adaptive octree grids with two-point nonlinear fluxes // Russian Journal of Numerical Analysis and Mathematical Modelling. — 2013. — Т. 28, № 3. — С. 267–288.

[2] Демьянович Ю. К. Всплесковые разложения на неравномерной сетке // Труды СПбМО. — 2007. — Т. 13. — С. 27–51.

[3] Демьянович Ю. К. Теория сплайн-всплесков. — СПб : Изд-во С.-Петерб. ун-та, 2013.

[4] Демьянович Ю. К., Дронь В. О., Иванцова О. Н. Об аппроксимации B_φ- сплайнами // Вестн. С.-Петербург. ун-та. Сер. 10. Прикл. матем. Информ. Проц. упр. — 2013. — С. 67–72.

[5] Демьянович Ю. К., Пономарев А. С. О реализации сплайн-всплескового разложения первого порядка // Численные методы и вопросы организации вычислений. XXIX. — 2016. — Т. 453. — С. 33–73.

[6] Лебедев А. С., Лисейкин В. Д., Хакимзянов Г. С. Разработка методов построения адаптивных сеток // Вычислительные технологии. — 2002. — Т. 7, № 3. — С. 29–43.

[7] Малла С. Вейвлеты в обработке сигналов. — М. : Мир, 2005.

[8] Сазонова Г. О. Построение адаптивной сетки для дискретного потока // Процессы управления и устойчивость. — 2016. — Т. 3(19), № 1. — С. 407–503.

Приложение 1

```
/// <summary>
/// builds adaptive net
/// </summary>
/// <param name=''iw''>index of u to start</param>
/// <param name=''ires''>index of res to start</param>
void grid(int iw, int ires, int adnetstop)
{
adnetres[ires] = w.wavData2()[iw];
adnetindex[ires] = iw;
int prev = iw;
int current = iw + 1;
ires++;
while (current != adnetstop)
{
short delta = abs(w.wavData2()[current] - w.wavData2()[prev]);
if (delta <= eps)
{
current++;
continue;
}
else
{
if (adnetres[ires - 1] != w.wavData2()[current - 1])
{
adnetres[ires] = w.wavData2()[current - 1];
prev = current - 1;
adnetindex[ires] = current - 1;
ires++;
}
else
{
```

```
adnetres[ires] = w.wavData2()[current];
prev = current;
adnetindex[ires] = current;
current++;
ires++;
}
}
}
if (adnetres[ires - 1] != w.wavData2()[adnetstop - 1])
{
adnetres[ires] = w.wavData2()[adnetstop - 1];
adnetindex[ires] = adnetstop - 1;
ires++;
}
adnetsize = ires;
}

/// <summary>
/// rebuild data by adaptive net
/// </summary>
/// <param name=''adnetstop''>size of adnet or midres</param>
/// <param name=''iflow''>index in res to start</param>
/// <param name=''iadnet''>index in adnet to start</param>
void agrid(int iflow, int iadnet, int adnetstop)
{
flowrestored[iflow] = adnetres[iadnet];
//iadnet++;
iflow++;
while (iadnet != adnetstop - 1)
{
short delta = abs(adnetindex[iadnet + 1]
- adnetindex[iadnet]);
if (delta == 1)
```

```
{
iadnet++;
flowrestored[iflow] = adnetres[iadnet];
iflow++;
continue;
}

double k = (double)(adnetres[iadnet + 1] -
adnetres[iadnet]) / (adnetindex[iadnet + 1]
- adnetindex[iadnet]);
for (int i = 1; i <= delta; i++)
{
flowrestored[iflow] =
(short)round(i * k + (double)adnetres[iadnet]);
iflow++;
}
iadnet++;
}
}


/// <summary>
/// find index in adnet
/// </summary>
/// <param name=''q''>index of initial flow</param>
int findKappaIndex(int q)
{
int i = 0;
while (i != adnetsize)
{
if (q > adnetindex[i] && q < adnetindex[i + 1])
{
return i;
```

```
}
i++;
}
return -1;
}

/// <summary>
/// decomposition
/// </summary>
/// <param name=''iw''>index in initial flow to start</param>
/// <param name=''adnetstop''>size of data</param>
void makeWaveflow(int iw, int adnetstop)
{
int i = 0;
while (iw != adnetstop)
{
if (iw == adnetindex[i])
{
waveflow[iw] = 0;
iw++;
i++;
continue;
}
int j = findKappaIndex(iw);
waveflow[iw] = (short)round(w.wavData2()[iw] -
(double)w.SamplesPerS() / (double)(adnetindex[j + 1] -
adnetindex[j])*
((double)(adnetindex[j + 1] - iw) / w.SamplesPerS()
* w.wavData2()[adnetindex[j]] +
(double)(iw - adnetindex[j]) /
w.SamplesPerS() * w.wavData2()[adnetindex[j+1]]));
iw++;
}
```

```
}


/// <summary>
/// reconstruction
/// </summary>
/// <param name=''iw''>index in initial flow to start</param>
/// <param name=''adnetstop''>size of data</param>
void rebuildFlow(int iw, int adnetstop)
{
int i = 0;
while (iw != adnetstop)
{
if (iw == adnetindex[i])
{
flowrestored2[iw] = adnetres[i];
iw++;
i++;
continue;
}
int j = findKappaIndex(iw);
flowrestored2[iw] = waveflow[iw] + (double)w.SamplesPerS() /
(double)(adnetindex[j + 1]- adnetindex[j])*
((double)(adnetindex[j + 1] - iw) / w.SamplesPerS()
* adnetres[j] + (double)(iw - adnetindex[j]) /
w.SamplesPerS() * adnetres[j + 1]);
iw++;
}
}
```